

Sparsity-Constrained Transportation Problem

Annie I. Chen*, Stephen C. Graves†

February 2014

Abstract

We study the solution of a large-scale transportation problem with an additional constraint on the sparsity of inbound flows. Such problems arise in the management of inventory for online retailers that operate with many order fulfillment centers; each stock-keeping unit is typically kept in a limited number of these order fulfillment centers so as to reduce the operational overhead in the supply chain network.

We propose a computationally efficient algorithm that solves this sparsity-constrained optimization problem while bypassing complexities of the conventional integer programming approach. The effectiveness of the algorithm is demonstrated through a series of numerical experiments on synthetic data.

Keywords: inventory positioning, sparsity-constrained optimization, transportation problem, network design

1 Introduction

There has been a boost in the business of online retail in recent years. The surge in customer demand has led to an expansion of available products online, which in turn again contributes to growth in demand. In addition, due to the unique nature of the virtual platform, products no longer need to be stocked in brick-and-mortar stores to serve customers in the immediate geographic vicinity, but can be distributed across a set of order fulfillment centers and shipped directly on demand. The resulting supply chain networks, being large-scale and integrated, give rise to the need for novel modeling and solution approaches.

Several important problems in online retail inventory management have been addressed. For example, significant attention has been devoted to efficient fulfillment strategies (i.e., which fulfillment center to ship from when a new order is received) and stocking policies (i.e., how much inventory to stock at each fulfillment center). A review of related topics can be found in Johnson and Whang (2002), Swaminathan and Tayur (2003), and Agatz et al. (2008).

The problem of interest in this paper is *inventory positioning*, a decision that precedes considerations for fulfillment and stocking level. Given the demand of products at each

*Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, anniecia@mit.edu.

†Sloan School of Management, Massachusetts Institute of Technology, sgraves@mit.edu.

geographic zone, the shipping costs between fulfillment centers and demand zones, and the capacity constraints at each fulfillment center, we would like to determine, for each product, which fulfillment centers it should be placed in so as to minimize the overall shipping cost. Without additional constraints, this problem could be modeled as a transportation problem and solved using well-known algorithms for multi-commodity network flow. However, due to the large number of distinct stock-keeping units (or items), it is often more practical to limit the number of fulfillment centers that carry each item, so as to reduce the operational overhead for managing the network. We refer to this as the *sparse-inbound constraint*, since it requires the vector of inbound flows in the transportation problem (which correspond to stocking levels at the fulfillment centers) to be sparse, i.e., to have a limited number of nonzero entries. To the best of our knowledge, this is the first systematic investigation of the inventory positioning problem under such conditions.

Sparsity-constrained optimization problems have been studied in the context of signal processing, particularly for applications where it is necessary to represent high-dimensional signals with sparse estimations that preserve most of the information while requiring much less memory space to store. Several optimization methods have been suggested, including the greedy approaches presented in Beck and Eldar (2012) and Bahmani et al. (2013). However, these algorithms are valid only for problems that do not have other constraints besides sparsity, and therefore cannot be directly adapted to our setting where additional network flow constraints need to be satisfied.

This paper presents an approach to the sparse-inbound transportation problem. The next section provides a mathematical formulation of the problem. The following section introduces a fast algorithm for finding near-optimal solutions. Its effectiveness is then demonstrated with numerical experiments. The final section concludes the work and briefly summarizes our ongoing research.

2 Problem Formulation

The sparse-inbound transportation problem can be formulated as a multi-commodity network flow optimization problem on a bipartite graph $G = (U \cup V, E)$. The nodes in the network are partitioned into disjoint sets U and V , which represent the order fulfillment centers and the demand zones, respectively. Every edge in E is directed from a node in U to a node in V . Let I represent the set of commodities (or items) that flow across the network. We assume that the following information is given:

- Cost c_{uv} : the cost for sending a unit from fulfillment center $u \in U$ to demand zone $v \in V$.
- Capacity l_u : the maximum total flow (the total amount of items) that each fulfillment center $u \in U$ is able to process.
- Demand z_v^i : the required outbound flow for each item $i \in I$ at each demand zone $v \in V$.
- Sparsity s_i : the maximum number of fulfillment centers that each item $i \in I$ is allowed to flow through.

Our goal is to find a set of cost-minimizing flows $x = \{x_{uv}^i\}$ and $y = \{y_u^i\}$, where x_{uv}^i denotes the flow of item i across edge (u, v) , and y_u^i is the inbound flow of item i at node u . In addition to the standard network flow constraints of flow conservation, edge capacity, and nonnegativity of flows, each inbound flow vector $y^i = (y_u^i)_{u \in U}$ also has to satisfy the given sparsity constraint of not having more than s_i nonzero entries. Mathematically, this optimization problem can be described as follows:

$$(\mathbf{P}) \quad \min_{x,y} \quad \sum_{(u,v) \in E} c_{uv} \sum_{i \in I} x_{uv}^i$$

$$\text{subject to} \quad \|y^i\|_0 \leq s_i \quad \forall i \in I \quad (\text{inbound flow sparsity}) \quad (1)$$

$$\sum_{i \in I} y_u^i \leq l_u \quad \forall u \in U \quad (\text{inbound capacity}) \quad (2)$$

$$y_u^i = \sum_{v \in V} x_{uv}^i \quad \forall u \in U, \forall i \in I \quad (\text{inbound flow conservation}) \quad (3)$$

$$z_v^i = \sum_{u \in U} x_{uv}^i \quad \forall v \in V, \forall i \in I \quad (\text{outbound flow conservation}) \quad (4)$$

$$x_{uv}^i \geq 0 \quad \forall (u, v) \in E, \forall i \in I \quad (\text{nonnegativity}) \quad (5)$$

$$y_u^i \geq 0 \quad \forall u \in U, \forall i \in I \quad (\text{nonnegativity}) \quad (6)$$

Limitations of the MIP approach

The complexity of problem (\mathbf{P}) arises mainly from the sparsity constraints (1). The conventional approach is to model such constraints as the following mixed integer program (MIP):

$$(\mathbf{P-MIP}) \quad \min_{x,y} \quad \sum_{(u,v) \in E} c_{uv} \sum_{i \in I} x_{uv}^i$$

$$\text{subject to} \quad y_u^i \leq M b_u^i \quad \forall i \in I, u \in U \quad (7)$$

$$\sum_{u \in U} b_u^i \leq s_i \quad \forall i \in I \quad (8)$$

$$\sum_{i \in I} y_u^i \leq l_u \quad \forall u \in U \quad (\text{inbound capacity})$$

$$y_u^i = \sum_{v \in V} x_{uv}^i \quad \forall u \in U, \forall i \in I \quad (\text{inbound flow conservation})$$

$$z_v^i = \sum_{u \in U} x_{uv}^i \quad \forall v \in V, \forall i \in I \quad (\text{outbound flow conservation})$$

$$x_{uv}^i \geq 0 \quad \forall (u, v) \in E, \forall i \in I \quad (\text{nonnegativity})$$

$$y_u^i \geq 0 \quad \forall u \in U, \forall i \in I \quad (\text{nonnegativity})$$

$$b_u^i \in \{0, 1\}$$

In this formulation, b_u^i are binary variables indicating whether or not item i flows through fulfillment center u , and M is a large positive constant (which can be set to, for example, the

sum of demands for all items in all demand zones). The constraints (7) ensure that $b_u^i = 1$ if $y_u^i > 0$, and (8) limits the total number of edges with positive flows.

Standard MIP solvers can be applied to obtain the optimal solution of **(P-MIP)**. However, due to the combinatorial nature of the problem, solving the MIP requires intensive computation and does not scale up to practical applications which may have hundreds of fulfillment centers and demand zones as well as thousands or even millions of items. It is therefore necessary to consider heuristic approaches that can find near-optimal solutions quickly.

3 The Algorithm

We now present an efficient two-stage algorithm that finds near-optimal solutions without using integer variables. We begin by explaining an important concept that frames the algorithm as a search algorithm; then, we present details about the two stages of the algorithm, followed by a discussion on trade-offs captured by the selection of parameters.

3.1 Conceptual description

The proposed algorithm is based on the following key observation about the set of feasible solutions of **(P)**: though not a convex set, the feasible set of **(P)** can be expressed as the union of convex sets (in particular, polyhedrons). Indeed, the sparsity constraint (1) is equivalent to requiring $N_i := |U| - s_i$ of the variables in the vector $y^i = (y_u^i)_{u \in U}$ to be 0:

$$\begin{aligned} & \|y^i\|_0 \leq s_i \quad \forall i \in I \\ \Leftrightarrow & \left(y_{u_1}^i = y_{u_2}^i = \dots = y_{u_{N_i}}^i = 0 \quad \forall i \in I \right) \text{ or } \left(y_{(u_1)'}^i = y_{(u_2)'}^i = \dots = y_{(u_{N_i})'}^i = 0 \quad \forall i \in I \right) \text{ or } \dots \\ \Leftrightarrow & \bigcup_{\{\tilde{U}_i, i \in I: |\tilde{U}_i| = N_i\}} \left(y_u^i = 0 \quad \forall u \in \tilde{U}_i, i \in I \right). \end{aligned}$$

Each statement in parenthesis above can replace (1) in **(P)** to result in a linear program (with a polyhedral feasible set) whose optimal solution satisfies the original sparsity constraint. We shall henceforth refer to such a linear program as a *sparse-LP*. For any given sparse-LP, we say that the node u is *inactive* for item i if the constraint $y_u^i = 0$ is present; otherwise, it is *active* for item i .

As we can see from the expression above, there are $\binom{|U|}{N_i}$ ways to choose the set \tilde{U}_i for each item i . Therefore, there are $\prod_{i \in I} \binom{|U|}{N_i}$ possible sparse-LPs (each corresponding to a parenthesized term above), and the union of their feasible sets is the feasible set of **(P)**. Moreover, the optimal solution of **(P)** is the overall optimal solution among all sparse-LPs.

Our algorithm is essentially a search algorithm over the optimal solutions of all possible sparse-LPs. Since the number of sparse-LPs increases exponentially with $|I|$, it is impossible to do an exhaustive search; however, as we shall see, our algorithm demonstrates that good heuristics could indeed go a long way.

3.2 The sparsify-improve algorithm

There are two consecutive stages in the algorithm, *sparsify* and *improve*:

1. In the *sparsify* stage, a sparse-LP is found by progressively “deactivating” nodes in a relaxed problem, i.e., adding constraints of the form $y_u^i = 0$ for some chosen u and i .

Specifically, the algorithm starts with the relaxed linear program of **(P)** without the sparsity constraint (1). In each successive iteration, it solves for the optimal solution of the current linear program, finds k_1 active edges y_u^i with the smallest flows, and deactivates each of them by adding the constraint $y_u^i = 0$ to the current linear program. This process is repeated until all items are left with no more than s_i active nodes and the sparsity constraint is satisfied.

Note that in general, due to the capacity constraints, not all sparse-LPs are feasible, and there is no guarantee that the sparsify stage will produce a feasible solution. However, we have found empirically that if **(P)** is feasible and the feasible set is not too small, then this search process will find a feasible sparse-LP with high likelihood.

2. Once a (not necessarily optimal) sparse-LP is found, the algorithm then enters the *improve* stage, in which it seeks to improve the solution by exploring other sparse-LPs that are obtained by swapping inactive nodes in the current sparse-LP with active nodes of the same item.

In particular, in each iteration, the algorithm checks up to k_2 inactive inbound edges, in decreasing order of the dual variable magnitude for the constraint $y_u^i = 0$ (or in increasing order of dual variable values, since the values are negative), to see if activating the edge results in an improvement. Dual variables are a natural choice for a heuristic rule, because they reflect the marginal cost of each constraint, i.e., the rate at which the optimal cost would change if the constraint $y_u^i = 0$ were relaxed to allow item i to flow across node u .

For each inactive edge selected for checking, the algorithm removes the constraint $y_u^i = 0$, thereby activating node u for item i and obtaining a relaxed problem. In order to satisfy the sparsity constraint, however, it is necessary to deactivate another node; we choose deactivate the node for the same item that has the smallest flow in the relaxed problem. The optimal solution is then updated by resolving the new sparse-LP.

The iteration ends once an improvement is found, or if there are no improvements found after checking k_2 inactive edges with the largest duals. In the former case, a new iteration is initiated; in the latter case, the algorithm terminates and returns the sparse solution that has been found.

We summarize this in pseudocode on the next page.

3.3 Parameter selection

There are two parameters for this algorithm: k_1 , the number of nodes to deactivate in each iteration of the *sparsify* stage, and k_2 , the maximum number of alternative sparse-LPs to consider in the *improve* stage.

Algorithm 1 The Sparsify-Improve Algorithm

Function definitions:

$sol(p) :=$ optimal solution of problem p (a vector of flows)
 $val(p) :=$ optimal value of problem p (the minimum cost)
 $argmin_i[k]\{y_i : i \in I\} :=$ indices of the k smallest elements

Input: sparse-inbound transportation problem **(P)**

Initialize:

$lp \leftarrow$ **(P)** without (1)

// Stage 1: Sparsify

while lp is not sparse **do**

$smallest_flows \leftarrow argmin_{(u,i)}[k_1]\{y_u^i : y_u^i \in sol(lp), (u,i) \text{ active in } lp\}$

for (u,i) in $smallest_flows$ **do**

 add constraint $y_u^i = 0$ to lp *// deactivate (u,i) for lp*

end for

end while

// Stage 2: Improve

$improving \leftarrow \text{true}$

while $improving$ **do**

$best_duals \leftarrow argmax_{(u,i)}[k_2]\{|dual(y_u^i = 0)| : y_u^i \in sol(lp), (u,i) \text{ inactive}\}$

$new_LPs \leftarrow \phi$

for $(u,i) \in best_duals$ **do**

$lp' \leftarrow lp$

 remove constraint $y_u^i = 0$ from lp' *// activate (u,i) for lp'*

$(u',i') \leftarrow argmin_{(u,i)}\{y_u^i : y_u^i \in sol(lp'), (u,i) \text{ active}\}$

 add constraint $y_{u'}^{i'} = 0$ to lp' *// deactivate (u',i') for lp'*

if $val(lp') < val(lp)$ **then**

$lp \leftarrow lp'$

 go to the top of the while loop

end if

end for

$improving \leftarrow \text{false}$

end while

Output: $sol(lp)$

The choice of k_1 determines the number of iterations in the *sparsify* stage. The algorithm begins with a relaxed linear program where all $|U| \times |I|$ nodes and items are active, but only $\sum_{i \in I} s_i$ are to be kept in the final sparse-LP. Deactivating the remaining $N := |U||I| - \sum_{i \in I} s_i$ nodes requires $\frac{N}{k_1}$ iterations, which decreases as k_1 increases. However, as k_1 grows, the search also becomes “coarser”, and the resulting sparse-LP would likely be less optimal, since fewer intermediate solutions are considered on the search path. Thus, there is a trade-off between optimality and computation time for the choice of k_1 . Opting for computation time seems to be the better approach, since the *sparsify* stage is only used for finding an initial sparse-LP, which would then be further refined in the *improve* stage. Therefore, we suggest scaling k_1 according to the problem size.

As for k_2 , the likelihood of finding an improved solution increases with k_2 , and therefore the *improve* stage is expected to terminate with fewer iterations. However, the running time of each iteration could also increase linearly with k_2 , since up to $2k_2$ linear programs are solved in each iteration. In addition, the “marginal likelihood” of finding a better solution decreases as k_2 grows large—in other words, loosely speaking, the best improvement possible is likely to have a large dual variable value, so if k_2 is “large enough” that it already includes the best improvement possible, further increasing it does not help with finding a better solution. Due to these complicated factors, there is no obvious choice for k_2 , and it should be selected based on the allowable computation time.

4 Numerical Experiments

In this section, we demonstrate the effectiveness of our algorithm by presenting numerical experiment results for problems of various sizes. As we shall see, the algorithm significantly outperforms the MIP approach in its capability for handling large-scale problems, finding near-optimal solutions in a much shorter amount of time.

4.1 Setup

In our experiments, the transportation networks are generated by placing nodes (order fulfillment centers and demand zones) uniformly at random on the unit square $[0, 1) \times [0, 1)$. To ensure feasibility, an edge is created to connect every warehouse to every demand zone. Characteristics of the network are determined as follows:

- The cost c_{uv} for edge (u, v) is set to be the Euclidean distance between fulfillment center u and demand zone v .
- The capacity l_u of each fulfillment center u is set to be a large number such that all sparse-LPs are feasible. In our case, all capacities were set to $2 \sum_{i \in I} \frac{\sum_{v \in V} z_v^i}{s_i}$, which is twice the total demand averaged across the number allowable fulfillment centers for all items.
- The demand z_v^i for each item i at each demand zone v is a randomly generated integer between 10 and 1000.
- The number of allowable fulfillment centers is set to $s_i = 5$ for every item.

The experiment is performed on problems with $|U| = 30$ fulfillment centers, $|V| = 100$ demand zones, and a varying numbers of items ranging from 1 to 64 in powers of 2. Parameters for the *sparsify* and *improve* stages are set to $k_1 = \lceil \sqrt{|I|} \rceil$ and $k_2 = 20$.

The program is written in Python and uses the Gurobi Optimizer. Experiments are executed on a machine with a dual-core 1.60GHz processor.

4.2 Results

For each value of $|I|$, the experiment is repeated on 10 randomly generated networks. For each problem instance, when possible, we also solve for the exact optimal solution using the MIP approach (**P-MIP**). The results for the Sparsify-Improve algorithm are compared with that for the MIP according to two performance metrics: computation time and optimality.

Table 1: Computation time and optimality of Sparsify-Improve and MIP

$ I $	Average computation time			Average optimality (%)
	MIP	Sparsify-Improve	S-I/MIP (%)	
1	0.917	1.523	166.09	2.10
2	5.46	4.013	73.50	3.35
4	17.26	11.01	63.78	4.74
8	90.13	39.68	44.03	5.41
16	376.15	111.97	29.77	6.35
32	997.63	338.17	33.90	5.63
64	-	1058.281	-	-

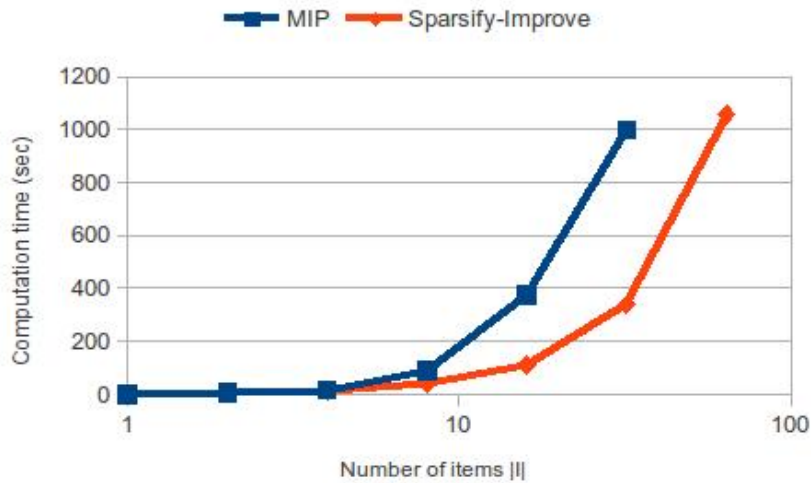


Figure 1: Computation time comparison of Sparsify-Improve and MIP

As we can see from the results summarized in Table 1 and Figure 1, the Sparsify-Improve algorithm produces solutions that are around 5% optimal on average, and it takes much less computation time than the MIP (the column labeled “S-I/MIP” is the ratio of the average computation time of the Sparsify-Improve algorithm to that of the MIP). Moreover, it also requires less computational resources than the MIP—for example, in the case where $|I| = 64$, the MIP program runs out of memory, and therefore no results are available for comparison.

5 Conclusion

In the previous sections, we presented a mathematical formulation of the sparse-inbound transportation problem, which is relevant for online retail inventory positioning problems with sparsity constraints on the number of fulfillment centers. We developed a two-stage algorithm for optimizing this model: the *sparsify* stage produces a sparse solution, which is then refined in the *improve* stage. The algorithm is based on heuristics, and it significantly outperforms a naive MIP approach, finding a near-optimal solution whose cost is around 5% optimal with much less computation time. Our findings make it possible to efficiently compute sparse inventory positioning plans, thus simplifying the management of large-scale online retail supply chain networks.

We are currently continuing our investigations on the sparsity-constrained inventory positioning problem. Our pursuits include a better understanding of the optimal solution structure, as well as theoretical analyses of the algorithm. Among other things, this could lead to more concrete performance guarantees, better guidelines for choosing algorithm parameters, and possibly even more effective algorithms for the sparse-inbound transportation problem that could be applied to generic sparsity-constrained optimization problems. We are also exploring the incorporation of additional constraints that reflect other operational considerations, as well as alternative models and algorithms for this problem.

References

- Agatz, N. A., Fleischmann, M., and van Nunen, J. A. (2008). E-fulfillment and multi-channel distribution — a review. *European Journal of Operational Research*, 187:339–356.
- Bahmani, S., Raj, B., and Boufounos, P. (2013). Greedy sparsity-constrained optimization. *Journal of Machine Learning Research*, 14:807–841.
- Beck, A. and Eldar, Y. C. (2012). Sparsity constrained nonlinear optimization.
- Johnson, M. and Whang, S. (2002). E-business and supply chain management: An overview and framework. *Production and Operations Management*, 11(4):413–423.
- Swaminathan, J. and Tayur, S. (2003). Models for supply chains in e-business. *Management Science*, 49(10):1387–1406.